Advanced Robot Control
Real-Time Operating System

Wojciech Domski

Chair of Cybernetics and Robotics,
Wrocław University of Science and Technology

Presentation compiled for taking notes during lecture

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Operating System (1/2)

What is an operating system?

Operating System (2/2)

It is a specific software that runs in kernel mode (supervised mode) [3].

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Wrocław University of Science and Technology

OS support features in Cortex-M3/M4 core (1/1)

Cortex-M3/M4 support following features to facilitate the implementation of an embedded OS [4]:

- **Shadowed stack pointer**
- **SysTick timer**
- **Supervisor Call (SVC) and Pendable Service Call (PendSV) exceptions**
- **Unprivileged execution level**
- **Exclusive accesses**

Outline

Real-Time Operating System

The RTOS is a special kind of an operating system which focuses on time. Two groups of RTOS can be specified:

- soft real-time operating system,
- hard real-time operating system.

Hard RTOS

Hard real-time operating system is a specific branch of RTOS which has to meet strict time constrains.

Soft RTOS

In contrast to Hard RTOS a Soft RTOS is an operating system where occasional violation of strict time regime is undesirable but will not lead to any permanent damage.

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Main Stack Pointer and Processor Stack Pointer

Figure: Stack pointer in a task [4]

Outline

Handler mode and Thread mode (1/1)

Figure: Operation states and modes [4]

Memory Protection Unit (1/2)

The MPU is a programmable device that can be used to define memory access permissions (e.g., privileged access only or full access) and memory attributes (e.g., bufferable, cacheable) for different memory regions [4].

Memory Protection Unit (2/2)

The MPU can be used to make an embedded system more robust .

Outline

Starting task with SVC

Figure: Initialization of a task [4]

Switching task with PendSV

Figure: Context switching [4]

Switching task during interrupt (1/2)

Figure: Improper task switching [4]

Switching task during interrupt (2/2)

Figure: Proper task switching [4]

Outline

Hardware Mutex support (1/1)

A common way to limit the access to a resource is a semaphore. It can be represented as a counter which holds a number of resource.

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Introduction

FreeRTOS is a real-time kernel on top of which embedded applications can be built to meet their hard real-time requirements [1].

Fatures (1/1)

- Abstracting away timing information
- Maintainability/Extensibility
- Modularity
- Team development
- Easier testing
- Code reuse
- Improved efficiency
- Idle time
- Power Management
- Flexible interrupt handling
- Mixed processing requirements

ISR dedicated API

ISR specific API

It is worthy of note that not all API can be called in ISR routines.

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Task states

Figure: Task switching [1]

Task creation

```c
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                const char * const pcName,
                uint16_t usStackDepth,
                void *pvParameters,
                UBaseType_t uxPriority,
                TaskHandle_t *pxCreatedTask );
```

```c
void Task( void *pvParameters ) {
  for(;;) {
  }
}
```

```c
vTaskStartScheduler();
```

Wrocław University of Science and Technology

Delaying task execution

```
void vTaskDelay( TickType_t xTicksToDelay );
```

```
void vTaskDelayUntil( TickType_t * pxPreviousWakeTime,
                      TickType_t xTimeIncrement );
```

```
TickType_t xLastWakeTime;
xLastWakeTime = xTaskGetTickCount();
```

Outline

Mutexes (1/2)

A mutex can be created using [2]

```
SemaphoreHandle_t xSemaphoreCreateMutex( void );
```

```
BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore,
    TickType_t xTicksToWait );
```

```
BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );
```

Mutexes (2/2)

During usage of mutexes a situation called *Deadlock* can arise.

Critical section

A critical section is a region of code surrounded with

```
taskENTER_CRITICAL();
// ...
taskEXIT_CRITICAL();
```

Outline

Queues (1/3)

Queues are normally used as **First In First Out (FIFO) buffers**, where data is written to the end (tail) of the queue and removed from the front (head) of the queue.

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
UBaseType_t uxItemSize );
```

```
BaseType_t xQueueSendToFront( QueueHandle_t xQueue,
const void * pvItemToQueue,
TickType_t xTicksToWait );
```

```
BaseType_t xQueueSendToBack( QueueHandle_t xQueue,
const void * pvItemToQueue,
TickType_t xTicksToWait );
```

Queues (2/3)

```
BaseType_t xQueueReceive( QueueHandle_t xQueue,
void * const pvBuffer,
TickType_t xTicksToWait );
```

```
UBaseType_t uxQueueMessagesWaiting( QueueHandle_t xQueue );
```

Queues (3/3)

Queue can be used as a **message box** where size of a queue is equal to 1.

```
BaseType_t xQueueOverwrite( QueueHandle_t xQueue,
const void * pvItemToQueue );
```

```
BaseType_t xQueuePeek( QueueHandle_t xQueue,
void * const pvBuffer,
TickType_t xTicksToWait );
```

Outline

Event groups (1/2)

Event group is a mechanism available in FreeRTOS which allows to suspend a task in a blocking state as long as a certain condition is not met.

Event groups (2/2)

```
EventGroupHandle_t xEventGroupCreate( void );
```

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
const EventBits_t uxBitsToSet );
```

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t
    xEventGroup,
const EventBits_t uxBitsToWaitFor,
const BaseType_t xClearOnExit,
const BaseType_t xWaitForAllBits,
TickType_t xTicksToWait );
```

Outline

Software timers (1/4)

Software timers can run a function (callback function) at a specific time or periodically.

Software timers (2/4)

```
TimerHandle_t xTimerCreate( const char * const pcTimerName,
TickType_t xTimerPeriodInTicks,
UBaseType_t uxAutoReload,
void * pvTimerID,
TimerCallbackFunction_t pxCallbackFunction );
```

```
BaseType_t xTimerStart( TimerHandle_t xTimer, TickType_t
    xTicksToWait );
```

```
BaseType_t xTimerStop( TimerHandle_t xTimer, TickType_t
    xTicksToWait );
```

Wrocław University of Science and Technology

Software timers (3/4)

```
void vTimerSetTimerID( const TimerHandle_t xTimer, void *pvNewID
    );
```

```
void *pvTimerGetTimerID( TimerHandle_t xTimer );
```

Software timers (4/4)

```
BaseType_t xTimerChangePeriod( TimerHandle_t xTimer,
TickType_t xNewTimerPeriodInTicks,
TickType_t xTicksToWait );
```

```
BaseType_t xTimerReset( TimerHandle_t xTimer, TickType_t
    xTicksToWait );
```

State machine for one shot sw timer (1/1)

Figure: One-shot software timer states and transitions [1]

State machine for auto-reload sw timer (1/1)

Figure: Auto-reload software timer states and transitions [1]

Outline

1. Operating system
   - OS support features
   - Real-Time Operating System
   - Shadowed stack pointer
   - Supervised and unsupervised mode
   - Context switching
   - Exclusive accesses
2. FreeRTOS
   - Introduction
   - Tasks
   - Mutexes
   - Queues
   - Event groups
   - Software timers
   - Heap management
3. Quiz

Heap implementations

In FreeRTOS a number of heap implementations is available:

- **Heap_1**
- **Heap_2**
- **Heap_3**
- **Heap_4**
- **Heap_5**

Quiz for videoconference (1/1)

Prepare yourself for a short test. Select the host of the meeting as the chat receiver. Do not send answers to everyone.
You will have 60 seconds for each question. When writing answer to the question. write down also the question number.
Question 0. What is your favourite colour? Answer 0. My favourite colour is blue.

Wrocław University
of Science and Technology

Quiz (1/1)

Calculate group number as the rest from dividing the Student ID number by 4.

Example

Student ID number is 123456, thus the group is 0.
Take last 2 digits from Student ID number (56) and calculate the rest from dividing by 4 (56 % 4 = 0).

Write down your name, Student ID number and group.

Literature (1/2)

R. Barry.
Mastering the FreeRTOS™ Real Time Kernel.
Real Time Engineers Ltd., 2016.

Amazon Web Services.
The FreeRTOS™ Reference Manual, API Functions and Configuration Options.
Amazon.com Inc., 2017.

A. S. Tanenbaum and H. Bos.
Moder Operating systems.
Pearson, 2014.

J. Yiu.
The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors.
Newnes, 2013.

Literature (2/2)